TDDI

Model Method: is responsible for calling TMDb. If the model does not exist yet, seam is use to test the code we wish we had.

Seams: a place where you can change apps behavior with out changing source code. It is useful for testing which isolates behavior of some code from that other code it depends on. Each seam enables just enough functionality for some specific behavior under test.

should_receive : uses Ruby's open classes to create a seam for isolation controller action from behavior possibly buggy or missing.

stub: similar to shuld_receive but not expectation

mock: "stunt double", used for behavior berification(did method get called)

RSpec: each spec should test one behavior. used seam to isolate that behavior. Determine what type of expectation will check the behavior. Make sure the test fails for the right reason. Add code until test is green.

Fixture: statically preload some known data into database tables. database wiped & reload before each spec

Pros and uses: truly static data, configuration info that never changes. Easy to see all test data in one place.

const : may introduce dependency on fixture data.

Factory: create only what you need pertest. Sets up helpers to quickly create objects with default attributes, as needed per-test.

Pros- keep tests independent: unaffected by presence of objects they don't care about.

Cons/reasons not to use: Complex relationships may be hard to set up but may indicate too-tight coupling in code!

BDDI

Behavior-Driven Design(BDD) : develop the features you wish you had to describe how app will work. Via cucumber, user stories become acceptance test and integration test.

- -ask questions about behavior of app before and during development to reduce miscommunication (Validation vs. Verification).
- -Requirements written down as user stories(lightweight description of how app is used)
- -Concentrates on behavior of app vs. implementation of app.

Test-Driven Development(TDD): step definitions for a new story, may require new code to be written. Write unit & functional tests for that code first, before the code itself. That is, write tests for the code you wish you had.

User Stories: 1-3 sentences in everyday language , that fits on 3X5 index card, written by/with customer. 3 phrase must be there. User story must by formulated as acceptance test before code is written.

- -why 3X5?
  - -Nonthreatening => all stakeholders participate in brainstorming.
  - -Easy to rearrange => all stakeholders participate in prioritization
  - -Since stories must be short, easy to change during development(often get new insights during development)

Smart Stories:

- -*Specific & Measurable:* each scenario testable, implies know good input and expected results exist.
  - Not example(UI should be user-friendly)
  - Example
    - -Given some specific starting condition s
    - -When I do X
    - -Then one or more specific things should happen
- *-Achievable* complete in 1 iteration, If cannot deliver 1 iteration, delivered subset of stories. If < 1 per iteration, need to improve point estimation per story.
- *-Relevant* “Business Value” Discover business value, or kill the story.
  - -Protect revenue
  - -Increase revenue
  - -Manage cost
  - -Increase brand value
  - -Making the product remarkable
  - -Providing more value to your customers.
- *-Time boxed:*
  - -Stop story when exceed time budget. Give up or divide into smaller stories or reschedule what is left undone.
  - -To avoid underestimating length of project
  - -Pivotal Tracker tracks velocity, helps avoid underestimate.

Backlog : user stories not yet completed. You have to prioritize most valuable items highest. Organized so they match SW releases over time.

Lo-Fi UI - Pen and paper. Tedious to do sketches, also less intimidating to nontechnical stakeholders => more likely to suggest changes to UI if not code behind it.
Storyboards - Need to show how UI changes based on user actions.Like scenes in a movie but not linear.

Spike

-Short investigation into technique or problem
-After spike done, code must be thrown away. Now know approach you want, write it correctly. Not all code branches merge into production.

Velocity: avg. number of points / week

Points: does not matter if velocity is 5 or 10, as long as team is consistent. Idea is to improve self-evaluation and suggest number of iterations for feature.

Pivotal Tracker:

- calculates velocity for team, managers user stories: Current, Backlog Ice box.
-Prioritize user stories by where place them in Current, Backlog, Icebox panels.
-When complete, move to done panel.
-Can add logical release points, so can figure out when a Release will really happen. Remaining points/velocity

Tracker Roles

- Developers don't decide when user stories completed.
-Product owner tries out the user story and then either hits “accept or reject”
    -accept, which marks user story as done
    -reject, which marks story as needing to be restarted by developer

Features

- user stories that provide verifiable business value to customer. Add agree box to checkout page. Worth points & therefore must be estimated

Chores

User stories that are necessary, but provided no direct, obvious value to customers. Find out why test suite is so slow. No points

Campfire: web-based service for password-protected online chat rooms.

Cucumber - describes behavior via features & scenarios behavior driven design. Tests from *customer*-friendly user stories. When install, it creates commonly used steps definitions.

Acceptance: ensure satisfied customers.
Integration: ensure interface between modules consistent assumptions, communicate correctly.
-meets halfway between customers and developer. can connect to real tests

User story: refers to single feature

Feature: > 1 scenarios that show different ways a feature is used.

Scenario: 3-8 steps that describe scenario

Step definitions: Ruby code to test steps.

Steps: step definitions via regular expressions. Regexes match English phrases in steps of scenarios to step definitions. Steps definitions (Ruby code likely used capture string)

5 Steps keywords

-Given : steps represent state of world before event: preconditions
-When: steps represent event, simulate user pushing a button.
-Then steps represent expected postcondition; check if true.
-And & But extend previous step

RSpec tests individual modules that contribute to those behaviors (test driven development)

Capybara simulates browser

-Can interact with app to receive pages.
-Parse the HTML
-Submit forms as a user would.

BDDII

Cucumber: magically maps 3X5 card user stories onto acceptance tests and integration test for the application.

Pitfalls: careless use of negative expectations.
Beware of overusing “Then I should not see”, many, may outputs are incorrect.
Include positives to check results.

GOOD

-User stories- common language for all stakeholders, including nontechnical(3X5)
(LoFiUI sketches and storyboards)
-Write tests before coding - validating by testing vs. debugging.

BAD

-difficult to have continuous contact with customers
-Leads to bad software architecture

RailsII

Unit Test Should be first

-Fast: run subset of test quickly since you'll be running them all the time
-Independent: no tests depend on others, so can run any subset in order
-Repeatable: run N times, get same result to help isolate bugs and enable automation
-Self-checking: test can automatically detect if passed no human checking of output
-Timely: written about the same time as code under test with TDD, written first.

RSPec

Domain-Specific Language for testing(DSL) small programming language that simplifies one task at expense of generality.

migration - script describing changes, portable across DB types. can identify each migration, and know which ones applied and when, can be create reversible. Can manage with version control. Automated == reliably repeatable. Create tools to automate.

-rails generate migration -> creates migration
-rake db:migrate -> apply migration

Create, Read, Update, Delete (CRUD)

Rails generates SQL statements at runtime, based on on your Ruby Code

Create(save/create)

-Must called save or save! to save changes to DB
-'!' version throws exceptions if operation fails.
-combines new and save
-aquire a primary key
-behaviors inherited from active record::based

Read(where,find)

-class method where selects objects based on attributes.

Update(update attributes)

-Modify attributes, then save object

Delete(destroy)

- destroy is an instance method
-delete, which doe not trigger lifecycle callbacks(so avoid it)
-once an AR object is destroyed, you can access but not modify in memory object.

Model

-Subclassing from ActiveRecord::Base connects a model to the database , provides crude operations.
-Database table name derive from models name
-Database table column names are getter & setters for model attributes. Do not modify instance variables.

MVC

Model: methods to get manipulate data
Controller: get data from Model, make available to view

View: display data, allow user interaction

Forms

creating a resource usually takes 2 interactions

-new: retrieve blank form

-create: submit filled form

1. Identify the action that serves the form itself
2. Identify the action that receives submission
3. Create routes, actions, views for each

• Form elements' `name` attributes will appear as keys in `params[]`

• Helpers provided for many common elements

RASP

-**R**ead the error message. Really read it.

• **A**sk a colleague an informed question.

• **S**earch using StackOverflow, a search engine, etc.– Especially for errors involving specific versions of gems, OS, etc.

• **P**ost on StackOverflow, class forums, etc.

– Others are as busy as you. Help them help you by providing minimal but complete information

Error

-The backtrace shows you the call stack at the stop point

-in the log, use debug,info, warn, error, fatal methods to control amount of logging

-Don't use puts or printf. It has nowhere to go when in production

Ruby

-Interpreted

-Object Oriented: everything is an object, every operation is a method call on some object.

-Dynamically typed: objects have types, but variables don't

-Dynamic: add, modify code at runtime(metaprogamming), ask object about themselves.

OPP in Ruby

-All values are reference to objects

-Objects communicate via methods calls, also known as messages

-Each object has its own private state

-Every object is an instance of a class

-An objects class determine the objects behavior, how it handles methods calls, or class contains method definitions.

-contains explicit return

Naming
- -ClassNames use UppeCamelCase
- -Mothods& variables use snake_case
- -CONSTANTS(scoped) & GLOBAL not scope
- -symbols: like immutable strings whose values is itself.

a.b
- -a is the receiver to which you send the method call, assuming a will respond to that method.

Local variables
- - must be asking before use

Modules
- -reuse behaviors
- -high-level behaviors that could conceptually apply to many classes
- -Mechanism: mix-in

Classes
- -subclass reuses/overrides superclass methods
- -Mechanism: inheritance

Dry
- -do not repeat your self.
- -Refactor code so that it has a single place to do things.

Reflection
- -lets us ask an object questions about itself and have it modify itself

Metaprogramming
- -lets us define new code at runtime.

Lambas
- -a function, without a name

Expression-oriented
- -Most methods nondestructive

Architecture

Saas Infrastructure
1. communication: allow customers to interact with service
2. Scalability: fluctuations in demand during day + new services to add users raplidly
3. Dependability: service and communication available 24X7

Services on Clusters
- -Clusters: commodity computer connected by commodity Ethernet switches.

-More scalable than conventional servers and cheaper
-dependability via extensive redundancy
-Few operators for 1000s servers, careful selection of identical HW/SW, virtual Machine Monitos simplify
-push down cost by factors 3x to 8x
-Illusion of infinite scalability to cloud user

TCP/IP(Transmission Control Protocol / Internet Protocol)
-IP: no-guarantee, best-effort service that delivers packets from one IP address to another
-TCP: make IP reliable by detecting dropped packets, data arriving out of order, transmission errors, slow networks
-TCP ports allow multiple TCP apps on same computer.

Domain
-DNS is another kind of server that maps names to IP addresses

HTTP request
-request method GET,POST
-Uniform resource identifier URI
-HTTP protocol version understood by the client
-headers- extra info regarding transfer request

HTTP response
-Protocol version & status code
-Response headers
-Response body

Cookies
- customizations
-click tracking/ flow tracking
authentication

Rest
-Idea: uniform Resources Identifier(URI) names resources, not page or action. Self-contained: which resources, and what to do to it. Responses include hyperlinks to discover additional RESTful resources.
- A service in the SOA sense whose operations are like this a restful service
-Ideally, RESTful URIs name the operations.

Routes
-Maps HTTP, URI to controller action. Parses to convenient hash.
-Built in shortcuts to generate all CRUD routes

Design Patterns

-Design Patterns : promote reuse

-Archtectureal (macroscale) patterns
    -Model-view-controller
    -Pipe & Filter
    -Event-based
    -Layering Saas technology stack
-Computation patterns
    -Fast Fourier transform
    -Structured & unstructured grids
    -Dense linear algebra
    -Sparse linear algebra

UML
    -Unified Modeling Language UML: notation for describing various artifacts in OOP systems.
    - class diagram, showing class relationships and principal.

Solid OOP principals
    -Single Responsibility principle
        -A class should have one and only one reason to change. Each responsibility is an axis of change. Axis chances shouldn't not effect others. Define responsibilities and stick to them. Models with many set of behaviors. Big classes files are a tipoff of SRP violation.
    -Open/Closed principle
        -Classes should be open for extension, but closed for source medificaiton.
    -Liskov substitution
        -Subtypes can Substitutes for base types. Type/subtype != classes/ subclasses. With duck typing, substitutability depends on how collaborators interact with object.
    -Injection of dependencies
        -inject an abstract interface that a&b depend on. If not exact match adapter facade. Inversion: now b and a depend on interface vs. a dependingg on b.
    -Demeter principle

    Demter Principle
        - Replace method with delegate
        -Be ware of important events without knowing implementation details
        -Separate traversal from computation.

    SOLID Cavet

- Doing for statically typed languages, so some principles have more impact there. Avoid changes that require gratuitous recompiling. Use judgment: goal is deliver working & maintainable code quickly.